(12) UK Patent Application (19) GB (11) 2 354 092 (13) A

(13) Date of A Publication 14.03.2001

(21) Application No 9921410.8

(22) Date of Filing 11.09.1999

(51) INT CL[7]
G06F 17/30

(52) UK CL (Edition S)
G4A AEX AFL A12C A12D

(56) Documents Cited
WO 99/63454 A1 IE 910003450 A US 5367619 A
PC Magazine Vol.16, No.11, June 10 1997, ISSN 0888-8507, p219(7)

(58) Field of Search
UK CL (Edition R) G4A AEC AEX AFL AKBX
INT CL[7] G06F 9/445 11/00 17/24 17/27 17/30
ONLINE: COMPUTER EPODOC JAPIO WPI INTERNET

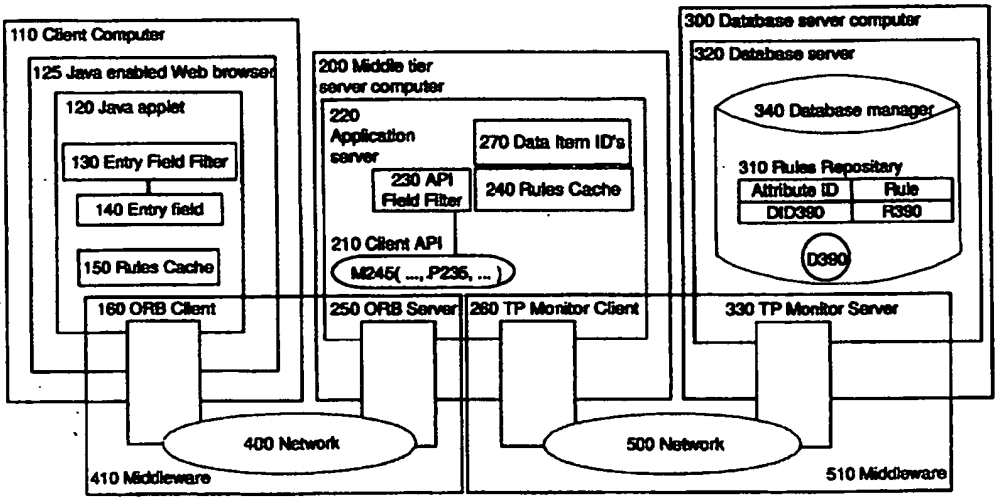
(71) Applicant(s)
International Business Machines Corporation
(Incorporated in USA - New York)
Armonk, New York 10504, United States of America

(72) Inventor(s)
Paul McDaid
Keith J Peters
Emer MacDowell

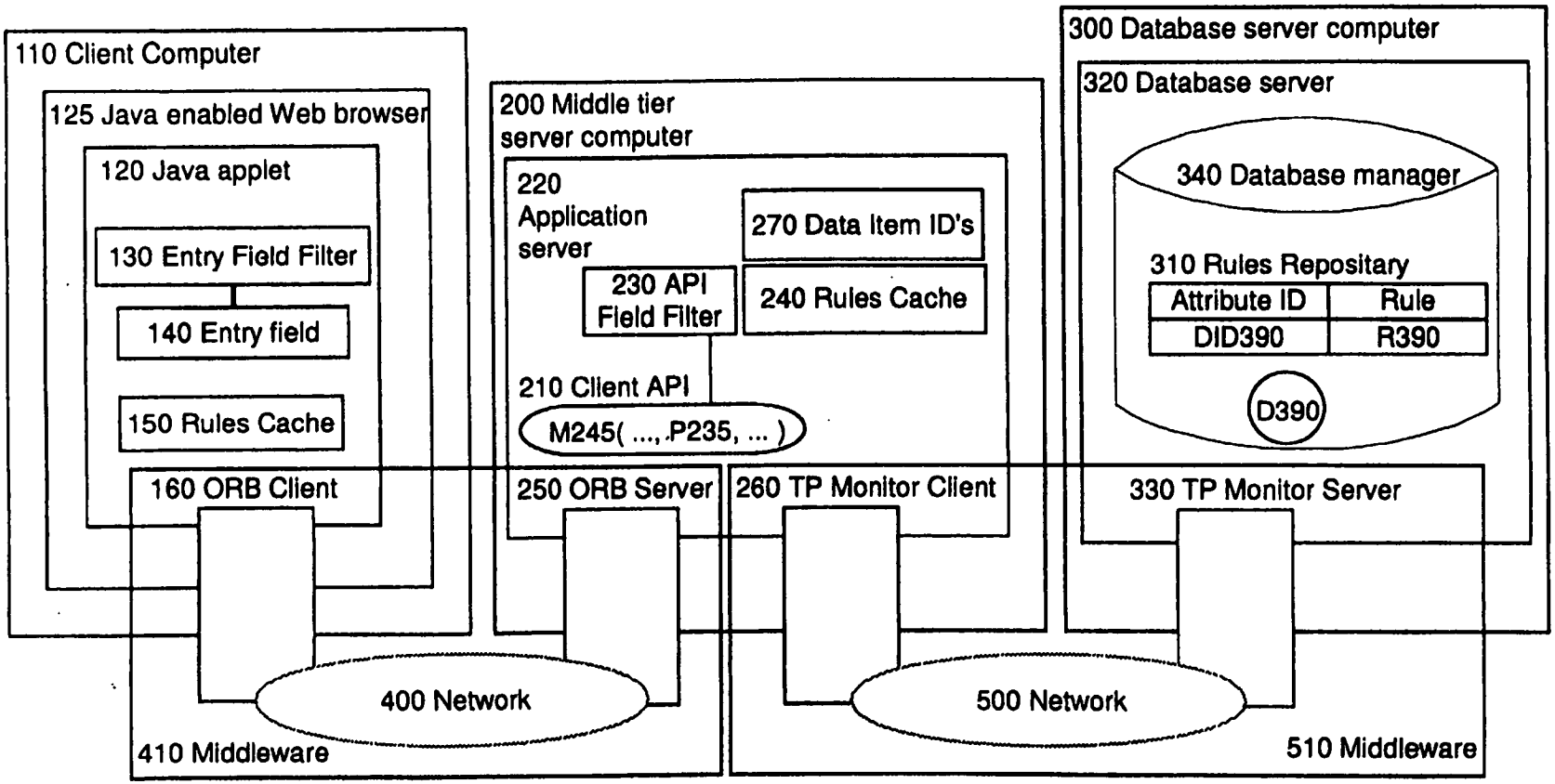
(74) Agent and/or Address for Service
IBM United Kingdom Limited
Intellectual Property Department, Mail Point 110,
Hursley Park, WINCHESTER, Hampshire, SO21 2JN,
United Kingdom

(54) Abstract Title
Filtering data in a user interface

(57) In a distributed system, a rules storage mechanism 150 may obtain rules from a second rules storage mechanism 310. A rule associated with a particular data type may then be provided from the first storage mechanism in response to an entry filter request. The entry filters are associated with input fields of given data types which provide information to one or more applications. The mechanism may be a Java class instantiable from an applet 120 running on a client 110. The system may involve one or more middle-tier computers 200.



GB 2 354 092 A


110 Client Computer
125 Java enabled Web browser
120 Java applet
130 Entry Field Filter
140 Entry field
150 Rules Cache
160 ORB Client
200 Middle tier server computer
220 Application server
270 Data Item ID's
230 API Field Filter
240 Rules Cache
210 Client API
M245(...,.P235, ...)
250 ORB Server
260 TP Monitor Client
300 Database server computer
320 Database server
340 Database manager
310 Rules Repositary
Attribute ID
Rule
DID390
R390
D390
330 TP Monitor Server
400 Network
500 Network
410 Middleware
510 Middleware

FILTERING DATA IN A USER INTERFACE

The present invention relates to a method and distributed system for pre-validating data inputted at the client interface of the distributed system according to a set of validation rules.

Systems where a client application is used to input data to a remote centralised or possibly distributed database across a network are known. A client application may support different types of client interface. One type may take the form of a Graphical User Interface (GUI) for gathering data from human end users. Another type may take the form of an Application Program Interface (API) through which other external applications may write data.

Although the types of client interface may differ, their underlying purpose - in the context of this invention - remains the same; to gather data from external sources for input to the distributed system for ultimate storage in a remote centralised or distributed database.

The way in which data is passed through the client interface will depend on its type. A GUI usually displays entry fields, each with associated prompts indicating to the user the required information. An API usually accepts data as input parameters whose meaning is described in an API reference manual. In the present specification, the term input field is used to refer to the mechanism by which an item of data is passed into a client interface. It corresponds to an entry field of a GUI or an input parameter of an API.

It is usual for data for a number of input fields to be gathered before any of the fields are processed by the system. In the case of a GUI the end-user usually strikes a function key or clicks on a user interface control to cause the entered information to be transmitted to a database manager which controls access to the database. In the case of an API, all the parameters of the API are provided before the API call is executed.

It will be seen that data should be validated before being stored in the database. In order to do this, the system designer is faced with a choice when designing the client application in that the client interface input fields are either hard coded to validate information being entered by the user; or unvalidated data is transmitted to the database manager, validated and, if there is an error, a message is sent back to the client application which advises the external system or end user that the content of one or more of the input fields is incorrect.

The problem with the first option is that validation still needs to be carried out by the server application prior to data being updated in the remote database to ensure information has been correctly transmitted across the network or that no corruption of stored data has taken place. This means that the system designer must keep duplicate sets of validation code synchronised in at least two separate applications resulting in a large systems maintenance overhead when the validation rules change. Moreover there is also a chance that improperly executed upgrades to a rule may result in the rule having one form in the client application and a separate form in the server application.

The problems in terms of GUI user friendliness of the second option are clear, in that users expect validation of the information to be instant and not to have to check back through their work for errors. Even if each input field's data is transmitted as it is entered there is still a user friendliness problem due to network latency which acts to slow the responsiveness of the user interface.

The problem in terms of API responsiveness of the second options are clear, the calling application must wait for the data to be transmitted to the database manager, validated and a response returned. The time spent transmitting the data and receiving a reply degrade responsiveness. This is a particular problem for API interfaces which have much more stringent response time requirements than GUI interfaces.

Another problem with the second option which applies equally to GUI and API interfaces is the fact that data which will fail validation will nonetheless still be sent for validation with error advice being returned. This adds to the traffic on the network causing reduced responsiveness and demanding an increase in network bandwidth.

It is an object of the present invention to mitigate such problems.

Accordingly, the present invention provides a mechanism according to claim 1.

An embodiment of the invention will now be described with reference to the accompanying drawing, which is a schematic drawing of a distributed client/server system for pre-validating data in a client interface. The preferred embodiment applies to the pre-validation of data entered via a GUI entry field and an API parameter.

As shown in the figure, a client computer 110 has a client application running on its operating system. In a preferred embodiment of the invention, the client application is a Java applet 120 running within a Java enabled browser 125. A middle tier server computer 200 has an

application server 220 running as an application on its operating system. The application server 220 provides an API 210 for use by external applications, for example, it may be used by the aforementioned client application 120. In the preferred embodiment of the invention, the application server 220 is a C++ program running on a Windows NT server (Windows NT is a trademark of Microsoft corp.). A database server computer 300 has a database server 320 running as an application on its operating system. In the preferred embodiment, the database server computer 300 is a high-powered mainframe computer running a database manager 340, such as DB2, on IBM's MVS operating system ("MVS" is a trademark of the IBM corp.). The Java applet 120, application server 220 and database server 320 together make up a known distributed system.

The Java applet 120 and the application server 220 intercommunicate using the services of middleware 410. This middleware facilitates the Java applet 120 on the client computer 100 calling the methods of the API 210 provided by the application server 220 running on middle tier server computer 200 including passing parameter data and receiving return data. In the preferred embodiment of the invention, the type of middleware used is an ORB comprising an ORB client 160 resident on the client 110 and an ORB server 250 resident on the middle tier server 200 communicating via a network 400.

Specifications and implementations of ORBs are known. The preferred embodiment of the solution uses the ORBIX ORB (ORBIX is a trademark of the IONA corp.) as the middleware. ORBIX conforms to the OMG ORB architecture called CORBA. The Object Management Group (OMG) is an international consortium of organizations involved in various aspects of client/server computing on heterogeneous platforms with distributed objects as is shown in the figure. The OMG has published a standard architecture - called CORBA (Common Object Request Broker Architecture) by which distinct applications; implemented in different languages; running as separate processes; on separate computers can inter-operate with each other in an object oriented programming fashion. Part of this standard architecture defines the responsibilities and interfaces of an ORB.

The application server 220 and the database server application 320 intercommunicate using the services of further middleware 510. This middleware facilitates the application server 220 on the middle tier server computer 200 calling the services of the database server 320 running on host database server computer 300 including passing parameter data and receiving return data.

In the preferred embodiment the middleware is a transaction processing monitor. Specifications and implementations of TP monitors are

known and the preferred embodiment comprises the CICS TP monitor (CICS is a trademark of IBM corp) comprising a TP monitor client 260 resident on the middle tier server 200 and a TP monitor server 330 resident on the database server 300 communicating via a network 500. The TP monitor hides the intricacies of the mechanisms involved in allowing an application on one computer execute a function of an application on another computer.

The database manager 340 includes a data item D390 which is one example of the many data items for which it receives values through its various client interface entry fields 140 and calls on the client interface API 210.

The Java applet 120 provides a GUI for use by end users. In the present example, the applet 120 includes a plurality of page objects which can be individually selected by any number of means, for example, when the applet is instantiated, the pages can be displayed superimposed with each page being selected by an associated tab which is always visible. Alternatively, the pages can be selected by browsing using a +- type tab.

In any case, each page object includes a constructor class which causes a number of entry field objects 140 (only one shown), sub-classed from the conventional Java entry field, to be instantiated. Each entry field object 140 includes within its a constructor, code to instantiate a filter object 130 which is associated with the entry field. The entry field object 140 provides to the filter 130 an attribute ID indicating the data type of the data item for which the entry field 140 is to accept from a user. (It should be seen that, unlike their associated entry fields, filter objects are not actually visible at run-time.) In the present example, entry field 140 is used to gather the value of the data item D390.

Parameter P235 of function M245 is an example of the one of many input fields provided by the application server API 210 and this parameter is also used to gather values of data item D390.

The system includes a rules repository 310 which persistently stores within itself at least one rule to determine if a particular value may be stored in a data item of a given type. The rules repository is located within the database server application 320 on the database server computer 300 and contains one rule for each data type. The rules are expressed according to a well defined syntax and grammar. In the preferred embodiment the rules include an indication of whether a data item is optional or mandatory, the data type of the data item and additional rules depending on the data type itself. For example, Integer data types may have a maximum and minimum allowable value specified,

Character String data types may have a maximum and minimum length and a permitted alphabet specified. In the figure, R390 represents the rule pertaining to the type of data stored in data item D390. Rules are linked to data items by means of an Attribute ID as each data item has a type and each type of data item has an associated Attribute ID and each rule is associated with an Attribute ID. In the figure, DID390 is the Attribute ID for data items of the same type as D390.

The system further includes a number of rules caches 150, 240. A rules cache runs within any application (120, 220) of the distributed system which has a client interface. A rules cache is a non-persistent store of some or all of the rules held in a rules repository. Furthermore both a rules repository and a rules cache provide APIs through which some or all of the rules they store may be queried by other applications or rules caches. In the present case this is carried out via middleware 510 and 410 respectively. In the preferred embodiment of the invention on start-up, the Java applet, running on the client computer, instantiates a rules cache 150 and another rules cache 240 is instantiated by the application server 220 running on the middle tier server computer when it starts up.

A rules cache contains a replica of the original rules, stored in the rules repository 310, which its calling field filters require. A rules cache may receive its copy of the rules directly from the rules repository or it may also receive its rules from another rules cache which has in turn received them from the rules repository.

In the preferred embodiment, the application server 220 includes a list 270 comprising Attribute ID's for each entry field 130 and each parameter of each method of the API 210. When started, the application server 220 instantiates the rule cache 240 and passes to it the list of attribute IDs 270. Application server 220 also creates each field filter 230 which is aware of the attribute ID of the data type for which it is to gather data.

It is the rules cache 240 which is responsible for connecting to the rules repository 310 and downloading rules from it, the applicable rules for all these Attribute ID's which were passed to it by the application server 220. This is facilitated by an API made available by the rules repository 310 and executable by the rules cache 240 by means of the middleware 510 already described.

In the preferred embodiment, the rules cache 150 receives its rules from rules cache 240. For this purpose rules cache 240 provides an API which may be called by rules cache 150 using the services of the middleware 410 already described. Thus, in the preferred embodiment rules

cache 150 receives its rules from rules cache 240 which in turn receives its rules from the rules repository 310.

A rules cache does not, on creation, immediately go to the rules repository or another rules cache and replicate the validation rules. Instead the rules cache is first advised (through an API which it publishes) by its field filters 130 or by the application server 220 that it will in the future require one or more particular rules. The rules which it is advised of are identified by their attribute ID. The rules cache only goes to the rules repository or another rules cache when the first request to return one of these rules is made by a field filter 130, 230.

Thus, when the first request is made of rules cache 240 to retrieve the rule for a data item, the rules cache will be empty and so the rules cache retrieves via the middleware 510 and stores within itself the rules for all the data types of which it was advised by the application server 220. The rules which rules cache 240 retrieves are those in the data item ID list 270 that was passed to it by the application server 240 on its startup. It is likely that this will in fact include every attribute ID used throughout the system, and even if this were strictly not necessary, it will be seen that the application server 220 will generally be available for long periods of time and so will only rarely need to populate the rules cache 240. The network 500 across which the middleware 510 connects the middle tier and the database server is usually a reliable link and so the time required to populate the rules cache is quite deterministic. On the other hand the connection between the applet 120 and the middle tier may be across the Internet for example and as such network latency is of great concern here.

In the preferred embodiment the rules cache 150 is advised by each entry field filter 130 of the Attribute ID of the data item for which it pre-validates data. Rules cache 150 accumulates within itself a list of all such Attribute IDs until it receives (via its API) a request to get one of those rules - usually in response to a user entering data in an entry field. In response, the cache 150 then issues a request to rules cache 240 to get the rules for all Attribute IDs in its list and, having received these rules, returns the appropriate rule to the calling filter 130.

Once their respective rules caches are populated, the invention enables data items for both for application server 220 and in turn the Java applet 120 to be validated immediately data is entered into an entry field or a method call is made.

It will also been seen that once a user has filled in all the entry fields on a page, which input has been immediately validated, a single call to the API 210 using the entry field values as parameters can be made. This single call again reduces to a minimum the required bandwidth on network 400. The data arriving to the application server 220, across what may have been an unreliable link, can again be instantly validated using the filters 230 associated with the parameters of the method call, and if successful, the information can be inserted into or used to retrieve information from the database server.

Thus, it can be seen that a rules cache will only populate itself with rules when a rule is actually requested by a field filter or the application server. In this way requests for many rules can be effected in a single call across middleware 500 or 410 further reducing network traffic.

The field filter 130, 230 of each input field 140, P235 stores within itself the Attribute ID for the data type of the data item collected by that input field. It is the responsibility of the field filter 130, 230 to validate data entered into its associated input field according to the rule associated with the data type of the data item. It can locate this rule from the rules cache running in the same application using the Attribute ID of the data type.

It has been demonstrated that by caching rules within the application collecting data, the rules can be checked as data is entered into the input field. Thus, end user performance is ensured and network traffic is minimised.

It has been demonstrated that rules stored in one single location can be dynamically distributed around the applications of the system. Thus, there is no systems maintenance overhead involved in synchronizing separate sets of hard coded rules nor is there an opportunity for a given rule to differ between client interface and the database server.

It should be seen that, for simplicity, the present embodiment has been described using only one client computer and one middle tier server. Nonetheless, in practice a number of client computers will connect to one of a potential number of middle tier servers which in turn connect back to a database server. Although logically unique, the database server can in turn be distributed over a number of machines.

CLAIMS

1. A rules storage mechanism for a distributed system in which one or more applications are adapted to receive information from a plurality of input fields, each input field being of a given data type and having an associated entry filter, said rules storage mechanism being instantiable by an application and comprising:

means for storing a plurality of rules each associated with a respective data type;

means for obtaining from a second rules storage mechanism rules for storing in said storage means; and

means, responsive to an entry filter request, for providing a rule associated with an input field data type from said rule storage means.

2. A rules storage mechanism as claimed in claim 1 further comprising:

means for storing a plurality of identifiers, each associated with an input field data type; and wherein said rules obtaining means is adapted to obtain rules associated with said identifiers from said second rule storage mechanism.

3. A rules storage mechanism as claimed in claim 2 wherein said rule providing means is responsive to said storage means being empty to cause said rules obtaining means to obtain rules associated with all of said plurality of identifiers.

4. A rules storage mechanism as claimed in claim 1, 2 or 3 wherein said mechanism is a class instantiable from within an application running on a first computer system connected to a network and said rules obtaining means is cooperable with a client process running on said computer system to obtain said rules from said second rules storage mechanism located on a second computer system connected to said network.

5. A rules storage mechanism as claimed in claim 4 wherein said class is a Java class instantiable from within an applet running on a client computer system, said client is an object request broker client, said second computer system is a middle tier server and said second rules storage mechanism is a non-persistent rules cache.

6. An applet including the rules storage mechanism as claimed in claim 5, said applet comprising a plurality of entry fields each having an associated entry filter, each filter being responsive to a change in its

associated entry field's contents to validate said contents using a rule provided by said rules storage mechanism.

7. A rules storage mechanism as claimed in claim 4 wherein said application runs on a middle tier server system, said client is a transaction processing monitor client, said second computer system is a database server and said second rules storage mechanism is a persistent rules repository.

8. An application including the rules storage mechanism as claimed in claim 7, said application comprising an application program interface comprising a plurality of method calls, each having a set of parameters, each parameter of a set having an associated entry filter, each filter being responsive to a call on a method with one of whose parameters it is associated to validate a parameter value passed in said method call using a rule provided by said rules storage mechanism.

9. A distributed system including one or more client computers, one or more middle tier computers and a database server computer interconnected via a network, said client computer including a means for running an applet according to claim 6.

10. A distributed system including one or more client computers, one or more middle tier computers and a database server computer interconnected via a network, said one or more middle tier servers including a means for running an application according to claim 8.

11. A method operable in a distributed system in which one or more applications are adapted to receive information from a plurality of input fields, each input field being of a given data type and having an associated entry filter, said method comprising the steps of:

obtaining from a remote rules storage mechanism a plurality of rules each associated with a respective data type;

storing said rules in a storage means; and

responsive to an entry filter request, providing a rule associated with an input field data type from said rule storage means.

12. A computer program product comprising computer program code stored on a computer readable storage medium for, when executed on a computing device, obtaining, storing and providing rules, the program code comprising the rules storage mechanism of claim 1.

The Patent Office

INVESTOR IN PEOPLE

Application No: GB 9921410.8 **Examiner:** Julyan Elbro
Claims searched: 1 to 12 **Date of search:** 31 March 2000

Patents Act 1977
Search Report under Section 17

Databases searched:

UK Patent Office collections, including GB, EP, WO & US patent specifications, in:

UK Cl (Ed.R): G4A (AEC, AEX, AFL, AKBX)

Int Cl (Ed.7): G06F 9/445, 11/00, 17/24, 17/27, 17/30

Other: ONLINE: COMPUTER EPODOC JAPIO WPI Selected Internet Sites

Documents considered to be relevant:

Category	Identity of document and relevant passage		Relevant to claims
E, A	WO 99/63454 A1	COLLEGENET see especially pages 24 and 25.	
A	US 5367619	EATON see abstract and fig. 4.	
A	IE 910003450 A	PETTIT see claim 1, figs. 2a and 2b, and related text.	
X	PC Magazine Vol. 16, No. 11, June 10 1997, W. R. Stanek, "VBScript 2.0 and JavaScript 1.2: scripting is the key to interactive Web pages.", ISSN 0888-8507, p219(7), especially the JavaScript section.		1 to 12

X	Document indicating lack of novelty or inventive step	A	Document indicating technological background and/or state of the art.
Y	Document indicating lack of inventive step if combined with one or more other documents of same category.	P	Document published on or after the declared priority date but before the filing date of this invention.
&	Member of the same patent family	E	Patent document published on or after, but with priority date earlier than, the filing date of this application.